Table of Contents

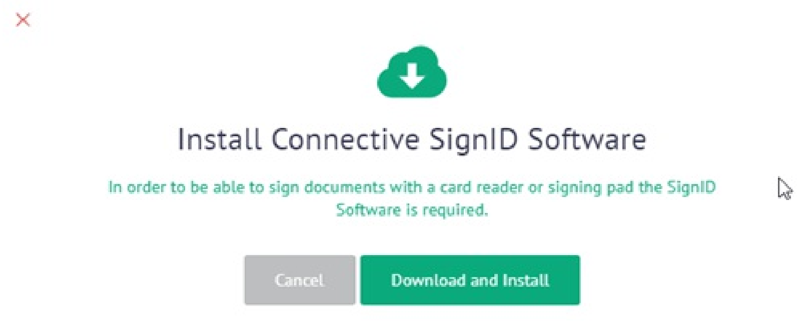
Installation Documentation

In this Installation Documentation you'll learn how to install **SignID** from within eSignatures on Windows and macOS.

Table of Contents

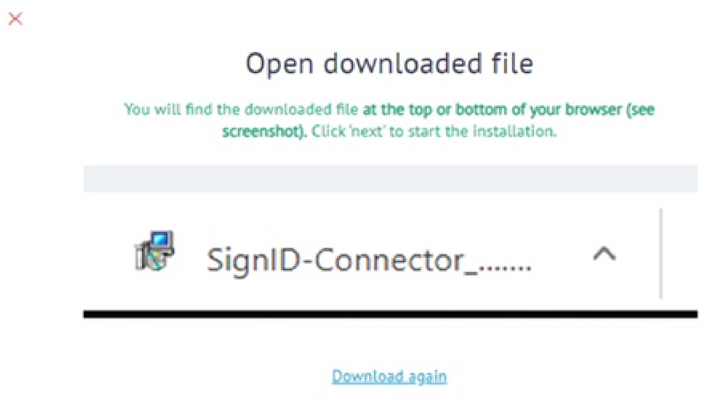
How to install SignID on Windows?

- Click **Start signing** in a document that requires eID, BeLawyer or Biometric signing. When the SignID software is not installed you are prompted to install it.
- Click **Download and Install** to start the download.


Install Connective SignID Software

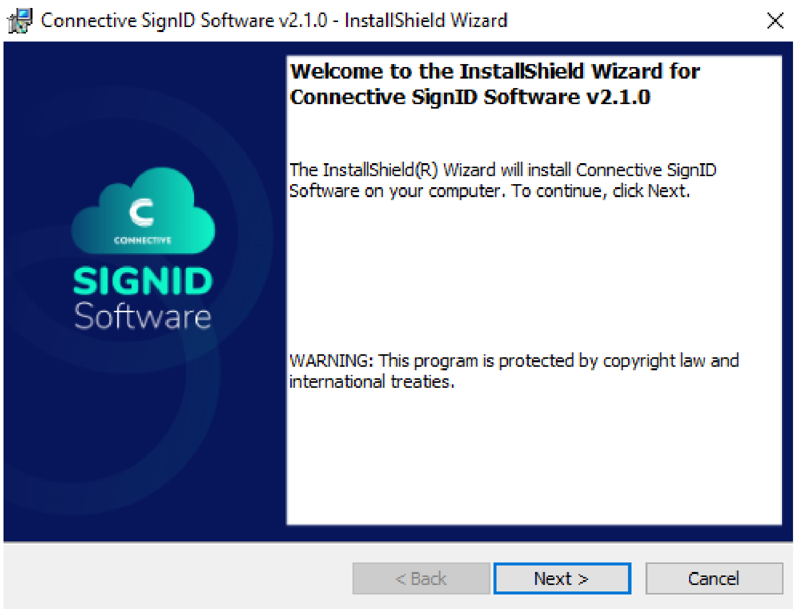
In order to be able to sign documents with a card reader or signing pad the SignID Software is required.

- Depending on the browser you're using, you'll find the downloaded file at the bottom or top of the windows.
- Click the downloaded file to start the installation.


Open downloaded file

You will find the downloaded file at the top or bottom of your browser (see screenshot). Click 'next' to start the installation.

- In the Installation Wizard, click **Next**.

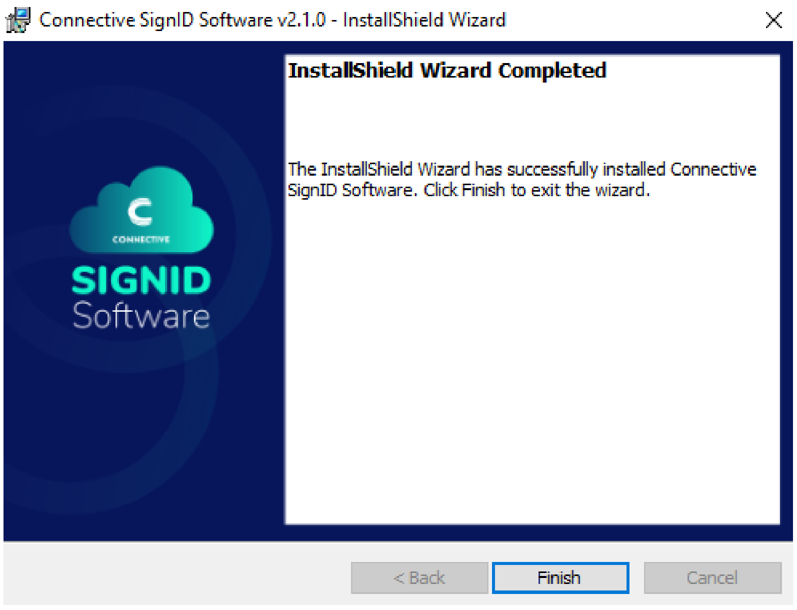

Connective SignID Software v2.1.0 - InstallShield Wizard

Welcome to the InstallShield Wizard for Connective SignID Software v2.1.0

The InstallShield(R) Wizard will install Connective SignID Software on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

- Then, click **Install**.
- If prompted, select **Yes** to allow this app to make changes to your device.
- When the installation is complete, click **Finish**.

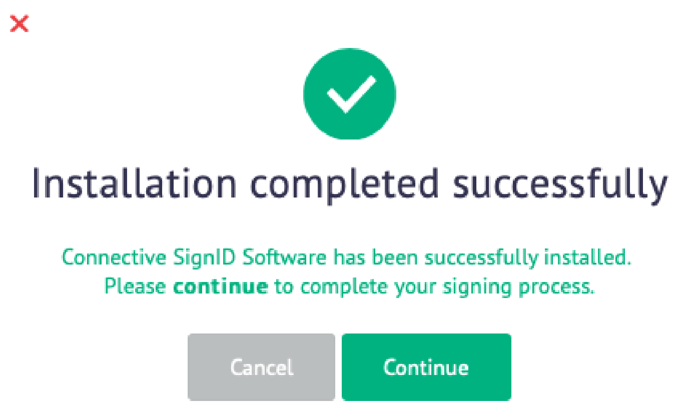

Connective SignID Software v2.1.0 - InstallShield Wizard

InstallShield Wizard Completed

The InstallShield Wizard has successfully installed Connective SignID Software. Click Finish to exit the wizard.

- Now click **Continue** to continue the signing process.

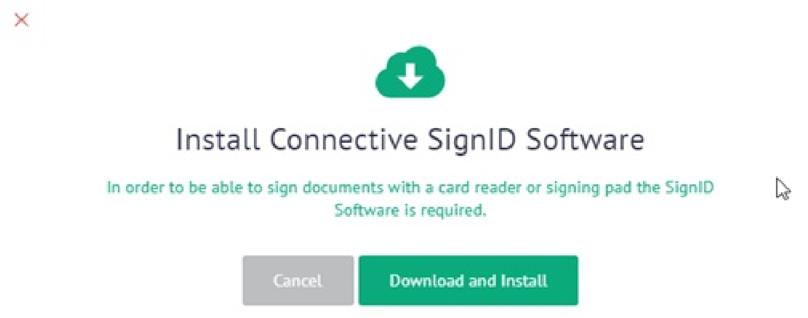
Attention: when using Mozilla Firefox, you need to close it and restart it after installing SignID software. Otherwise you will keep being prompted to install SignID.


Installation completed successfully

Connective SignID Software has been successfully installed.
Please **continue** to complete your signing process.

How to install SignID on macOS?

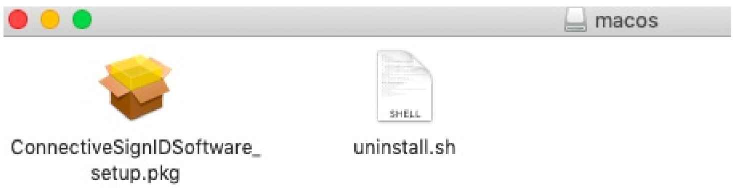
- Click **Start signing** in a document that requires eID, BeLawyer or Biometric signing. When the SignID software is not installed you are prompted to install it.
- Click **Download and Install** to start the download. If prompted, select **Save file**.


Install Connective SignID Software

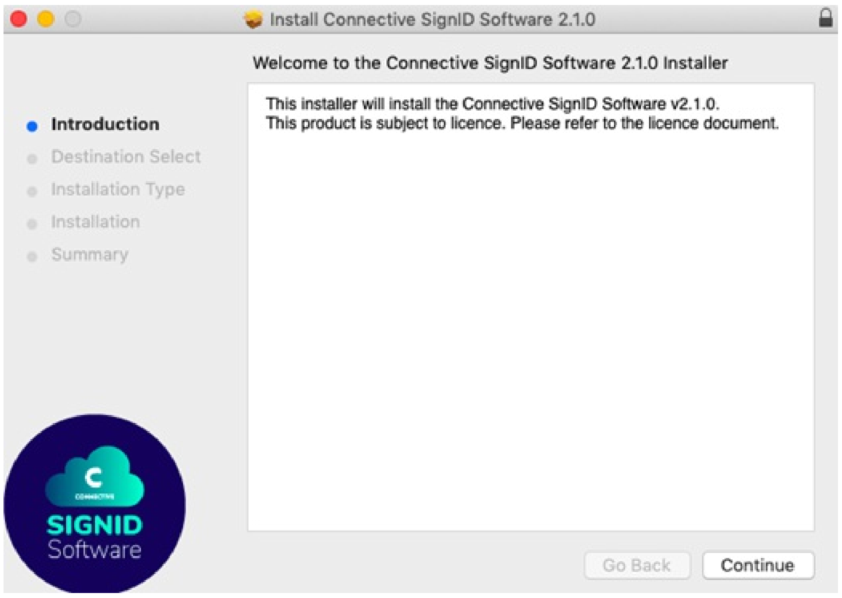
In order to be able to sign documents with a card reader or signing pad the SignID Software is required.

- The file is downloaded to the **Downloads** folder.
- Click **Downloads** in the Dock.
- Then double-click **SignID-Connector.dmg** to unpack it. This may take a few seconds.
- If you don't have the necessary admin rights on your computer then you'll to follow these steps as well:
  - Go to System Preferences.
  - Open Security and Privacy.
  - On the General tab, tap the lock at the bottom left of the screen.
  - Enter your password to unlock Security and Privacy.
  - Allow Connective software to be installed.

- In the folder that opens, double-click **ConnectiveSignIDSoftware_setup.pkg** to start the installation.



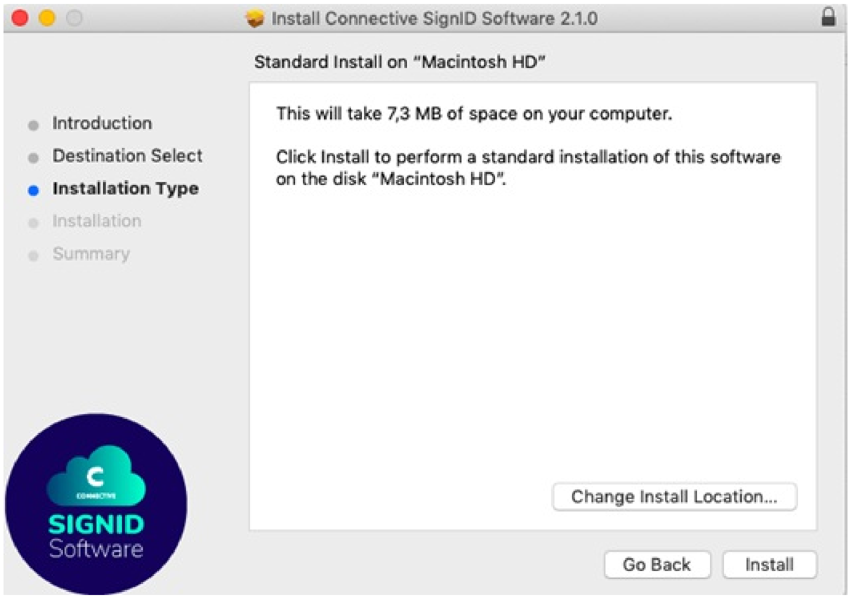
- In the installation screen, click **Continue**.


Install Connective SignID Software 2.1.0

Welcome to the Connective SignID Software 2.1.0 Installer

This installer will install the Connective SignID Software v2.1.0.
This product is subject to licence. Please refer to the licence document.

- Then click **Install**. It is recommended not to change the install location.
- If prompted, enter your account password, and then click **Install Software**.

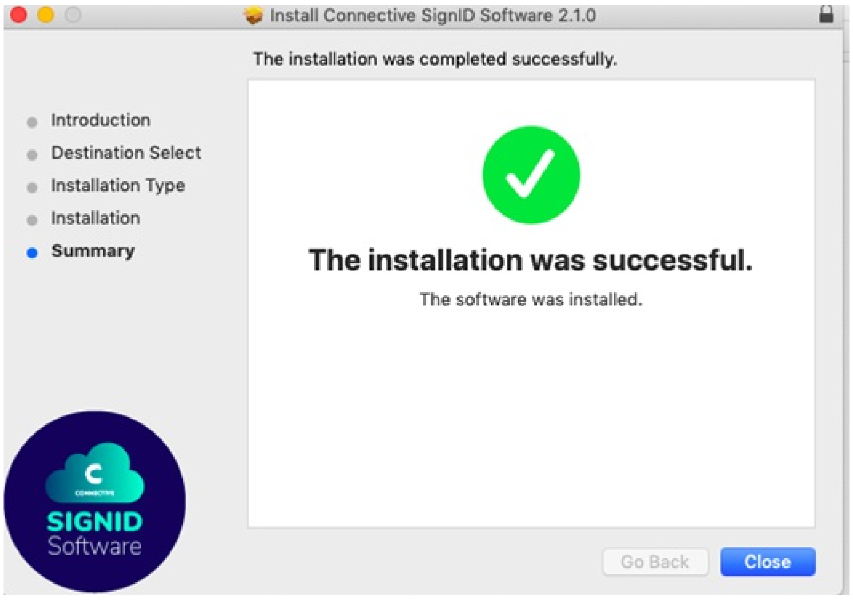


Standard Install on "Macintosh HD"

This will take 7,3 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk "Macintosh HD".

- When the installation is complete, click **Close**.

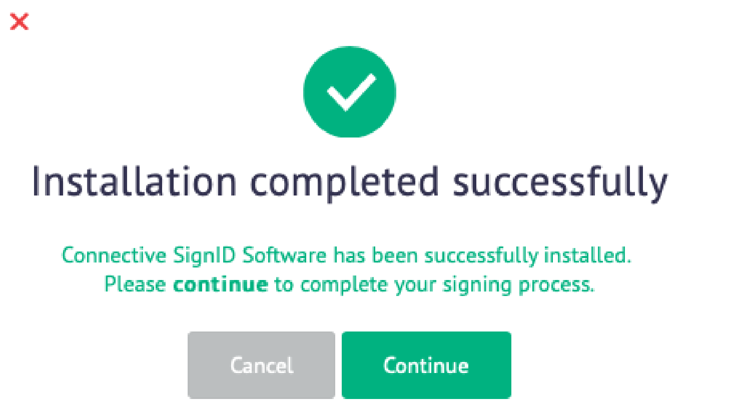


The installation was completed successfully.

The installation was successful.

The software was installed.

Attention: when using Mozilla Firefox, you need to quit it and restart it after installing SignID software. Otherwise you will keep being prompted to install SignID.

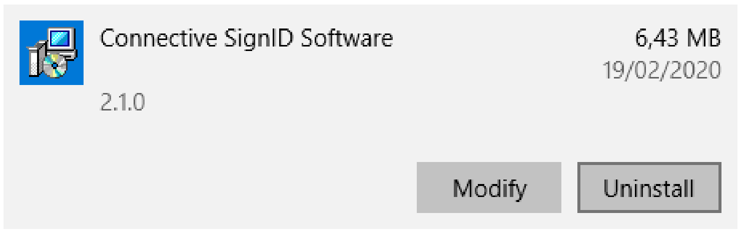


Installation completed successfully

Connective SignID Software has been successfully installed.
Please **continue** to complete your signing process.

How to uninstall SignID from Windows?

- Open **Apps and features**.
- Search for **Connective SignID Software** in the list.
- Select it and click **Uninstall**.



Connective SignID Software 6,43 MB
19/02/2020
2.1.0

- Make sure to restart your computer after uninstalling.

How to uninstall SignID from macOS?

- In **Finder**, go to **Applications** > **Utilities** > **Terminal**.
- Double-click **Terminal** to open it.
- Drag and drop the **uninstall.sh** file you find in the .dmg file to the Terminal, and click **Enter**. **Tip**: the .dmg file is the one you've downloaded to install the SignID software.
- When prompted, enter your password.
- The Connective SignID software will now be uninstalled.

Integration Documentation

This Integration Documentation describes how to integrate the **Connective Plugin Wrapper** into your own web application.

The new Connective Plugin Wrapper is modular and backwards compatible. It allows allows your web application to communicate with **SignID** software with the following aims:

- Connect to smart card readers to read out data
- Connect to smart card readers to compute signatures
- Connect to biometric signing pads to compute signatures

Table of Contents

0. Modules Overview

The Connective Plugin Wrapper consists of the following modules:

MODULE NAME	DESCRIPTION	REQUIRED?
Wrapper (main module)	Main module containing all relevant *interfaces* and the PluginApiFactory.	Required
SignID	SignID *implementation* of the main Wrapper module interfaces.	Required
Legacy-interop	Drop-in legacy replacement for the old Wrapper.	Optional

Notes

- The SignID module is required as this is currently the only module that implements all available features using the SignID software as its native component.
- The `legacy-interop` module must *only* be used if you're using an existing implementation based on our old Wrapper. The `legacy-interop` provides a fully backwards compatible drop-in replacement layer for the old `connective-plugin-wrapper` code. The interop module itself includes the old wrapper as a dependency since it is still required in order to provide the mobile implementation, which was not re-written. The legacy-interop module is fully optional and if no legacy support is required it is recommended to not include it.

1. Install the modules or include them in your html file

To install the modules:

- Include the pre-built files you obtained from Connective in `build/production` via a `<script>` tag, if you're not using any packaging tools like `webpack` or `rollup`.

  In this case, the library is exposed via the `ConnectiveWrapper` global variable.

Or

- Drop the package inside your node_modules dir (*Not recommended*).

2. Retrieve a PluginApi instance

Before you can use the `connective-wrapper`, you must first retrieve a `PluginApi` instance from the `PluginApiFactory`.

If the platform you're using is not supported - because of an outdated Operating System and browser combination for instance - the `PluginApiFactory` will be unable to create an instance and will throw a `PlatformNotSupportedException`.

To retrieve a PluginApi instance, follow the example below:

```
'use strict';

import { PluginApi, PluginApiFactory, Exceptions } from 'connective-wrapper';

const pluginApiFactory = PluginApiFactory.getInstance();
let pluginApi: PluginApi = null;
try {
    // note that we're also passing along a "gclOptions" object in our extraOptions dict
    // the gclOptions object is used by the t1t implementation module which uses t1c-lib-js
(https://github.com/Trust1Team/t1c-lib-js) library for the native layer
    // it should match the GCLConfigOptions interface as defined in that library.
    pluginApi = pluginApiFactory.createPluginApi({
        activationToken: '<activation token goes here>',
        extraOptions: {
            gclOptions: {
                gwOrProxyUrl: '<distribution service url goes here>',
                osPinDialog: true
            }
        }
    });
} catch(e: Error) {
    if (e instanceof Exceptions.PlatformNotSupportedException) {
        // current platform is not supported...
    }
}
```

Note that an activationToken must be entered in the indicated location. To obtain the activationToken, you need to do a REST call to the Distribution Service endpoint. This GET request must include a header containing "apiKey" as name, and the actual apiKey as value. The value of the apiKey should have been communicated to you during onboarding.

The apiKey will be exchanged for a valid JWT, which is the activationToken that must be entered. This token, which has a limited lifetime, will be forwarded to the front-end, which in turn will call the Distribution Service to obtain the URL to install the plugin for instance, or to check which card types or devices you are allowed to use.

Important: the apiKey must be kept secret, and ***must never*** be exposed to the front-end!

Also note that in the example above, we make use of the `extraOptions` object which includes a `gclOptions` entry. This optional entry is specifically used by the SignID module.

3. Initialize the PluginApi instance

Once the `PluginApi` instance has been created, it must be initialized.

The initialization verifies the presence of the native SignID software. If the native SignID software is missing, an error message will be displayed in which you will find a download link to retrieve the SignID software installer.

Note that the `initialize` method, like almost all other methods, is async and returns a `Promise`!

Also note that you can provide an extra configuration object to the initialize method. The one provided through the initialize method will be merged with the one that was used when retrieving the instance using the `PluginApiFactory` (in step 2). This is entirely optional however.

To initialize the PluginApi instance, follow the example below:

```
'use strict'

import { PluginApi, PluginApiFactory, Exceptions } from 'connective-wrapper';

const pluginApi: PluginApi = pluginApiFactory.getInstance();

pluginApi.initialize({ activationToken: '....' })
    .then(() => {
        // good to go. PluginApi instance is initialized and the native SignID software was detected
    })
    .catch((e: Error) => {
        if (e instanceof Exceptions.SoftwareRequiredException) {
            console.error(`Native SignID software could not be detected. Download at ${e.downloadUrl}`);
        } else if (e instanceof Exceptions.ActivationRequired) {
            // no activation token was provided OR the provided one was not (or no longer) valid
        }
    });
```

4. List connected card readers

Once the `PluginApi` instance has been retrieved and initialized, you will be able to list the connected card readers.

There are two ways to do this:

- You can either list all card readers and find the one you need:

```
'use strict'
// simply list all card readers. Array will be empty if there are not card readers.
pluginApi.getCardReader()
    .then((connectedReaders: CardReader[]) => {
    })
```

- Or you can let the Wrapper filter out card readers that contain a specific card:

```
'use strict'

import { BeID, BeLawyer } from 'connective-wrapper';

// list all card readers that contain a BeID card. Array will be empty if there are no readers containing a
BeID card.
pluginApi.getCardReader(BeID.cardInfo)
    .then((connectedReaders: CardReader[]) => {
    });

// OR list all card readers that contain a BeLawyer card. Array will be empty if there are no readers
containing a BeLawyer card.
pluginApi.getCardReader(BeLawyer.cardInfo)
    .then((connectedReaders: CardReader[]) => {
    });

// and so on...
```

Note that all card readers can tell you if they have a card inserted or not: If they have a card inserted, they will provide a `CardInfo` object containing an `atr` and `description` value with which the inserted card can be identified.

The Wrapper currently comes with 2 built-in supported cards, BeID and BeLawyer. Both provide a `CardInfo` object which can be used to detect the respective card type.

5. Read out data with a card reader

Once you've selected the required card reader by means of the `getCardReader` method, you need to create a `CardHandler` for the card that is present in the selected `CardReader`.

The `CardHandler` contains the actual implementation to handle whichever card is inserted and differs from one card to another.

Note: Don't worry about the different implementations, the `PluginApi` instance deals with this for us.

The data that is available on a card also depends on the card type. Just like the `cardInfo` object, both `BeID` and `BeLawyer` namespaces within the wrapper expose `DataKindHints` which can be used to retrieve a specific set of data on the card. The `readData` method will return a dictionary that contains a data result for each `DataKindHint` that was understood by the handler. All other `DataKindHints` that are not understood are ignored.

To create a `CardHandler` follow the example below:

```
let cardReader: CardReader;
let cardHandler: CardHandler;

pluginApi.getCardReader(BeID.cardInfo)
    .then((cardReadersWithBeID) => {
        cardReader = cardReadersWithBeID[0];
        cardHandler = pluginApi.getCardHandler(cardReader);
    })
    .then(() => {
        return cardHandler.readData(cardReader, BeID.DataKindHints.PersonalInformation)
    })
    .then((dataResult: {[index:string]: any}) => {
        // sadly we have to do some magic "casting" over here... the generic getData method currently has no
way to statically know the data type
        // that will be returned. The modules have defined that when fetching BeID PersonalInformation
DataKindHint it will return an instance
        // of the BeID.PersonalInformation interface and we must trust the module to do so.
        const personalInformation: BeID.PersonalInformation =
dataResult[BeID.DataKindHints.PersonalInformation];
        console.log(personalInformation);
    })
    .catch((e: Error) => {
        // ...
    });
```

6. Compute signatures with a card reader

Computing signatures is similar to reading data, but instead of using the `DataKindHints` the `SignatureKindHints` are used.

To compute a signature, follow the example below:

7. Compute signatures with a signer pad

SignID software can also be used to communicate with signer pads.

The `SignerPad` mechanism works exactly the same as the card readers. The only difference is that a `SignerPadHandler` can only compute signatures and does not provide any data to read like a `CardHandler` does.

8. Using the Legacy-interop module

The Legacy-interop module works in the same way as the old Wrapper.

There is one small, but **very important** difference however:

In the new API you can provide a `PluginApiOptions` configuration object to the Plugin instance when initializing. This configuration object is used in the SignID module to receive the SignID-specific configuration. For SignID this will contain the gateway endpoint, a pkcs11 config, etc. and is essential.

The legacy-interop `PluginApi` implementation (`PluginApiInterop`) accepts this `PluginApiOptions` object in its constructor, while the original implementation did not.

Tip: for more information about the gclOptions object, see step 2.

API Documentation

The Connective Wrapper API documentation can be found here.